Code Review - Disaster Management System

Project Name: Disaster Management System
Reviewed By: Aditya Nandkishor Shingne
Date: 31st January 2025

---

1. Code Overview

This script is designed to send disaster alerts via email to users based on their registered location. The main components include:

- A user database containing email addresses and locations.
- Functions for location validation, email sending, and alert distribution.
- A command-line interface to input disaster details.

---

2. Annotated Code Review

2.1 User Database

```
users = {
    "user1": {"location": "Sunder Nagar", "email": "b22111@students.iitmandi.ac.in"},
    "user2": {"location": "Mandi", "email": "user2@example.com"},
    "user3": {"location": "Kullu", "email": "b22134@students.iitmandi.ac.in"}
}
```

Review:

- The user database is a dictionary storing user emails and locations.
- Consider allowing dynamic user registration instead of hardcoding users.

2.2 Location Validation

```
def validate_location(location):
    """Validates if the specified location exists in the user database."""
    valid_locations = {details["location"] for details in users.values()}
    return location in valid_locations
```

Review:

- This function efficiently extracts valid locations from the user database.

- It could return a more detailed response (e.g., suggesting closest matches for invalid inputs).

2.3 Email Sending Function

```
def send_email(email, subject, body):
    """Sends an email alert to the specified recipient."""
    sender_email = "sample@gmail.com"  # Replace with your Gmail address
    app_password = "sample password [16 letters]"  # Replace with your actual app password

    try:
        msg = MIMEText(body)
        msg['Subject'] = subject
        msg['From'] = sender_email
        msg['To'] = email

        with smtplib.SMTP_SSL('smtp.gmail.com', 465) as server:
            server.login(sender_email, app_password)
            server.send_message(msg)
        print(f"Email sent to {email}")

    except Exception as e:
        print(f"Failed to send email to {email}: {e}")
```

Review:

- The function is well-structured and uses smtplib for secure email transmission.
- Sensitive credentials (email & password) should be stored securely (e.g., environment variables or a config file).
- Consider logging errors to a file for better debugging instead of just printing them.

2.4 Alert Distribution

```
def send_alert(location, message):
    """Sends alerts to users in the specified location."""
```

```
    if not validate_location(location):
        print(f"Error: '{location}' is not a valid location.")
        return

    print(f"\nSending Alert for {location}: {message}")
    recipients = [details["email"] for details in users.values() if details["location"] == location]

    if recipients:
        for email in recipients:
            send_email(email, f"Disaster Alert: {location}", message)
    else:
        print("No users found in this location.")
```

Review:

- The function correctly checks if the location exists before sending alerts.
- Instead of printing errors, consider returning structured responses (e.g., return False, "Invalid Location").
- It might be beneficial to allow batch processing for multiple locations at once.

2.5 Script Execution

```
if __name__ == "_main_":
    print("=== Disaster Alert System ===")
    location = input("Enter affected location: ").strip()
    message = input("Enter alert message: ").strip()
    send_alert(location, message)
```

Review:

- Typo: "_main_" should be "__main__" to execute properly.
- Consider adding input validation (e.g., check if message is empty before proceeding).
- A graphical or web-based interface could improve usability.

---

3. Summary of Improvements

Area	Suggested Improvement
User Database	Support dynamic user registration
Location Validation	Suggest nearest valid locations
Email Security	Store credentials securely
Error Handling	Log errors instead of printing
Alert System	Allow batch processing for multiple locations
Script Execution	Fix if __main__ typo and add input validation

---

4. Conclusion

The code is well-structured and achieves its purpose effectively. However, improvements in security, error handling, and usability could enhance the system's robustness. Implementing these changes will ensure better performance and user experience.

Aditya Nandkishor Shingne
31st January 2025